Mitsuba 系列 2-移植 Mitsuba 矩阵向量计算库

Dezeming Family

2022 年 6 月 22 日

目录

一　命令行交互与基本介绍

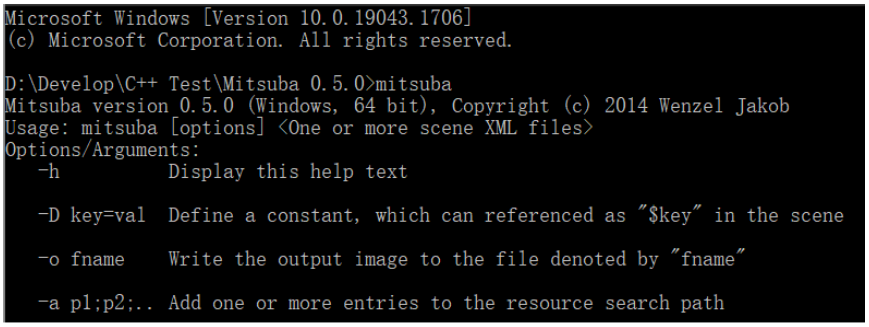
我们先对 [1] 的用户手册部分的一些解释和总结，其实初学者在了解一个渲染器时应该先学会使用，再研究代码，但作为稍有经验的开发者，直接接触代码或许会更直观。

图形界面交互较为简单，[1] 有教学视频的链接。

Windows 系统中，cmd 进入 mitsuba 目录下，输入命令并回车：

```
mitsuba
```

打印 Mitsuba 的基本信息。包括版本号、作者、版权以及一些操作命令。



调用 mitsuba.exe 的命令主要是用来调用渲染的。还有一些其他命令，是一些额外的程序，用来作为便利的工具或测试。mtssrv 与联网计算有关，可以在另一台主机上启动渲染。mtsutil 是功能函数调用，用户可以自己扩展一些功能，然后用该程序来调用。

二　构建基础工程

本节完成以后的代码见 [0 - InitialWorks]。

我们提供每个章节的源码，我们建议用户按照我们的要求来操作。

在某个最好没有中文的目录下新建一个 OurRender 文件夹,然后在该文件夹下放入我们提供的 SourceFiles，这里有我们全部的项目源码；然后我们再在 OurRender 目录新建一个 Build，该目录下用于生成工程。我们提供的源码都是需要用 CMake 来生成的，目标平台是 Visual Studio，应该没有平台限制，但我们倾向于大家使用 Visual Studio 2015 以及以上的版本。

我们需要在本地安装 Qt 开发环境，我的主机上安装的 Qt5 路径为：

```
D:/DevTools/QT5/5.7/msvc2015_64
```

如果你的目录不一样，那么你需要手动修改一下每个源码下的 CMakeLists.txt 中的 Qt 路径设置：

```
set(QT_PATH "D:/DevTools/QT5/5.7/msvc2015_64" CACHE PATH "qt5 cmake dir")
```

我们的初始工程只需要 Qt 这一个工具，后面当我们需要其他库的时候会逐步添加，所以用户不必担心一开始就要安装一大堆东西。CMake 也不需要额外的配置，直接选好目录然后 Configure 就好，如果出错说明 Qt 目录安装有问题，对照着 CMakeLists.txt 修改一下就好了。

成功生成以后，我们就进入下一个环节。

三　向量、点和法向量、矩阵的移植

本节完成以后的代码见 [1 - VectorAndMatrix]。

这些类和功能都是定义在头文件的，没有对应的 cpp 文件。虽然我更倾向于使用 OpenCV 或者 GLM 里的库和功能（更加完善和强大，资料也更多），但毕竟我们前面提到过尽量不增加过多别的库，所以我们选择移植 Mitsuba 自己的功能类。一个功能完善的矩阵向量计算库是制作一个渲染器的非常重要的基础。

3 1　向量、点和法向量

我们本节要把 Mitsuba 的 point.h、vector.h 和 normal.h 这三个文件里的内容移植到我们自己的 InitialWorks 里，得到新的工程为 VectorAndMatrix。

Mitsuba 有一个自己的名称空间，我们也需要命名一个名称空间。由于我们通篇代码都是在“抄袭”Mitsuba，我也不想再加上版权声明了，能看到该源码的用户肯定都是知道它来自于 Mitsuba 的，如果读者想要在自己的源码里使用 Mitsuba 的代码，务必加入 Mitsuba 的版权声明。

Mitsuba 的名称空间用下面的宏来包括，其实这个宏就是 namespace：

```
1 MTS_NAMESPACE_BEGIN
2 MTS_NAMESPACE_END
```

我们给自己的渲染器取一个名字，在这里我取名为 Dinodon（意为链蛇属蛇类，因为我当前正好养了一只漂亮的赤链蛇，就用它来做渲染器的名字了）。

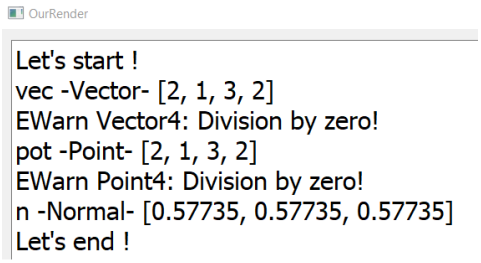
我们定义了一个基于单例模式的调试输出器 DebugText 类，在 MainGUI 目录下。该类的调用非常简单，使用宏即可，TextDinodonS 表示打印输出字符串，TextDinodonN 表示打印输出数字：

```
1 TextDinodonS(text) // 打印字符串
2 TextDinodonN(text, num1) // 打印字符串加一个数字
3 TextDinodonN2(text, num1, num2) // 打印字符串加两个数字
4 TextDinodonN3(text, num1, num2, num3) // 打印字符串加三个数字
5 TextDinodonN4(text, num1, num2, num3, num4) // 打印字符串加四个数字
6 TextDinVector(text, vec) // 打印一个Vector对象
7 TextDinPoint(text, pot) // 打印一个Point对象
8 TextDinNormal(text, norm) // 打印一个Normal对象
```

text 是我们要显示的前缀字符串。其实最后面这三个宏可以直接定义为一个，因为都是使用了各自类别的 toString() 方法，这里是为了好区分。

本节移植好的代码见 Vector.h、Point.h 和 Normal.h。类的声明在 Mitsuba 中放到了 fwd.h 文件里，我们也建立了一个对应的 Fwd.h（我比较喜欢文件名大写）。我们删除了与 Stream 有关的内容，出错和警告的调试信息也改为了 TextDinodonS 打印。

本节代码编译运行以后会输出显示以下内容：



3 2　矩阵

矩阵类分别定义在了两个头文件，一个是 matrix.h，另一个是 matrix.inl，其中.inl 是 Mitsuba 作者从 JAMA 库中移植过来的。我们把这两个头文件都搞到一个文件里，文件名为 Matrix.h。

移植的过程和前一小节是一样的，所以不再赘述。本节代码输出结果是：

```
Matrix Test start !
m_ivt -Matrix- Matrix4x4[
  -0.233333, -0.1, 0.466667, 0.0333333;
  -0.6, -0.4, 0.2, 0.8;
  0.166667, 0.5, -0.333333, -0.166667;
  0.566667, 0.1, -0.133333, -0.366667
]
I -Matrix- Matrix4x4[
  1, -8.9407e-08, -5.96046e-08, 1.19209e-07;
  1.19209e-07, 1, -5.96046e-08, 2.98023e-08;
  1.19209e-07, -7.45058e-08, 1, 5.96046e-08;
  1.19209e-07, -1.19209e-07, 0, 1
]
Matrix Test end !
```

将矩阵与矩阵的逆封装，就能得到变换类 Transform，这和 PBRT 是一致的。由于我们还需要对 Ray 做变换，所以需要提前移植 Ray 类。鉴于把 Ray 和 Transform 全都移植好工作量较大，故本文暂时到此为止。

四　本文小结

我们在本文给了两个代码工程，分别是 [0 - InitialWorks] 和 [1 - VectorAndMatrix]。大家可以尝试一下 Matrix 里面的一些矩阵计算操作，熟悉一下 Mitsuba 的库，这也是为我们后面使用奠定基础。

本文虽然较短，但希望大家也能自己移植，实际操作一下，这样就更能加深印象。在 PBRT 中，虽然我没有给出源码，而且笔风混乱，但也有不少读者坚持读到了第十本以后的内容，并且将源码移植成功，这让我很受感动。有些读者也指出了一些错误和问题，尤其是我在文中给出的代码片段中有些定义的变量之后一直没有用到（无效的冗余代码），于是，为了更方便读者学习，本系列将会全程提供源码，但仍旧希望读者根据自己的能力和兴趣来自己操作一遍。

参考文献

[1] https://www.mitsuba-renderer.org/docs.html